IBM Developer for z Systems – for ISPF Developers

Software Analyzer/Code Review
IBM Trademarks and Copyrights

© Copyright IBM Corporation 2008 through 2019.

All rights reserved – including the right to use these materials for IDz instruction.

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates.

This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM’s sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

IBM, the IBM logo, the on-demand business logo, Rational, the Rational logo, and other IBM Rational products and services are trademarks or registered trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.
The IDz Workbench Curriculum

- Module 1 – IDz Terms, Concepts and Navigation
- Module 2 – Editing Your COBOL Programs
- Module 3 – Analyzing COBOL Programs
- Module 4 – Remote Systems – Connect, Navigate and Search
- Module 5 – Remote Systems – Dataset Access and Organization
- Module 7 – MVS Subprojects – Organizing PDS Members and SCM Checkout
- Module 8 - The Data Tools – SQL Code/Test and DB2 Table Access
- Module 9 - Debugging z/OS COBOL Applications

Optional Modules

- IDz/Endevor Integration Through CARMA
- BMS & MFS Map Editors
- zUnit – Unit Test
- Code Coverage – Test quality feature
- Code Review – Application quality feature
- Menu Manager – Integrate ISPF REXX Execs and CLISTs
- Web Services – SOA development
Course Assumptions

1. You know ISPF and have used it for at least two years, doing production z/OS work in COBOL, PL/I or Assembler
   - Note that all of the workshops in this course are in **COBOL** – although files exist that are Assembler, PL/I, REXX and other languages for you to experiment with – time permitting

2. You have:
   - No experience with Eclipse or IDz
   - Some experience with PC tools
     - You have used MS-Windows applications for at least one year
   - IDz installed and running on your workstation at version 8.0 or later

   - Note that all ISPF discussion/examples and screen captures assume IBM-installed ISPF product defaults – not any 3rd party or custom Dialog Manager applications you may have installed on your mainframe
UNIT

The IDz Workbench

Topics:

- Overview of Code Review Feature/Function
- Custom COBOL Code Review Development
- Running Code Review in Batch on z/OS
Software Analyzer (Code Review) - Introduction

“Electronic desk-checking” that provides a means for you to enforce shop development standards and coding best practices

- Available for COBOL, PL/I and Java programs:
  - Opened Interactively:
    - Remote Systems Explorer
    - Local Workstation projects – including a z/OS library (PDS) downloaded to a local project
    - MVS SubProjects
  - Run in batch via JCL:
    - Especially applicable to supporting Continuous Integration and DevOps

- Easy to use:
  - Context-menu accessible – available from within Edit

- Easy to setup:
  - Create custom rule sets configuration based on in-the-box COBOL and PL/I rules
  - More requirements (from customers) are encouraged/gladly accepted

Highly customizable:
- In-the-box rules customizable through Preferences
- Out-of-the-box rules can be added through Java/Eclipse plugins

- Considerations:
  - Programs run through Code Review must be syntactically error-free
  - Can run reports and export findings on standards compliance and trends
Code Review - Three Options/Workflows

Interactive –
- Review a single program at a time:
  - During Edit
  - ... or from the Context Menu – from Local Project

Local Folder –
- Still interactive mode, but can Code Review a PDS/library at a time
- If the library has been copied down to a local workspace you can select and Code Review multiple programs

Batch/JCL –
- Review programs in one or more PDS/libraries at a time... or... Code Review an entire library
- Three Code Review Severity levels return as MVS Job Step Completion Codes ...
  - Allowing you to conditionally run other job steps – as part of Continuous Delivery/DevOps
Software Analyzer (Code Review) – Options and Extensibility features

- Code Review provides a number of static code analysis rules:
  - In-the-box: 52 for COBOL, 36 for PL/I rules, > 200 Java rules
  - A Java API + Wizard to create and import your own completely custom Code Review rules

- COBOL Code Review provides 18 Program Metrics

- There are multiple output report formats (XML, HTML, PDF, CSV)
  - IDz’s Code Review can integrate or feed other software or application analytic products in this space

- You can do program Baselining with Code Review:
  - The analytic findings are identified incrementally through point-in-time archives – yielding far fewer false positives

- Code Review provides a Post Processing script capability:
  - For custom actions you would like to automate upon discovery of coding rules/standards violations – often reports to kick off
Running Software Analyzer (Code Review) – Interactive Example

From within COBOL or PL/I Edit
- Right-Click
- Select Software Analysis => Your code review rules

Integration Test is a pre-defined “rule set”
Running Software Analyzer (Code Review) – Review Results

- Single program results

  Click this Red X to delete the results (and associated source tags)

- Each of the collapsed rule indicator allows you to hyper-link to the statement in the source program
**Setting up Software Analyzer (Code Review) – Process**

1. Define one or more base Rule sets
   - Select Rules
   - Choose Severity

2. Define parameter-driven rules
   - Specify container
   - Define rule(s) and their parameters

3. Add parameter-driven rules to the ruleset(s)

4. Run rules against:
   - A COBOL or PL/I program open in the editor
   - Or a library containing programs

5. Validate exceptions

6. Modify source

7. Repeat steps 4 ➔ 6

---

1. Optionally refine the ruleset
   - Add/Delete out-of-the-box rules
   - Code Custom Code Review Rules in Java

2. Run the rules in batch on z/OS
   - Run using a Baseline
Software Analyzer (Code Review) – Invoking from a Context Menu

You can drag an entire PDS to a Local Project and run Code Review against all of the programs at once.

And you could drag multiple libraries to a Local Project and run Code Review against all of the files in the workspace.

You can select multiple specific programs to run Code Review against in a Local Project.
Create a Ruleset – steps – Code Review – Access the Configurations Wizard

- Open **WARDRPT.cbl** in the editor
- Right-click and select:
  - Software Analyzer  >  Software Analyzer Configurations…
Steps - Code Review / Create a Ruleset

- Select Software Analyzer and Click: the New launch configuration icon
- Name your Ruleset ➔
- Select the Rules tab

- Expand COBOL Code Review
- Uncheck a few of the available default rules in the three ruleset categories:
  1. Naming Convention rules
  2. COBOL performance and run-time efficiency rules
  3. Code maintainability ("Program Structures") rules

- Click Close and Yes – to the "Save changes?" prompt

Notes
- You can return to Software Configuration, and modify your rule selection at any time.
- If you are using IDz v8.5 you will see different COBOL Code Review rules than this screen capture ➔
Steps – Review the Code in WARDRPT.cbl

- **Right-click and select:** Software Analyzer > Software Analyzer Configurations…
  - **COBOL RULES**
    - Or whatever you named your Ruleset

- **Note what happens:**
  - Any statement that "breaks" a rule is:
    - Flagged
    - Hyperlinked

- Navigate around in the results a bit until you get the idea behind Code Review

- Click the Red X icon in the view, to delete the Code Review analysis
Review all of the COBOL Code in a Local Folder or Mainframe PDS

- From z/OS Projects:
- Right-click over your cobol folder and select: Software Analyzer → Software Analyzer Configurations… → COBOL RULES
  - Or whatever you named your custom Ruleset
  - You might want to run this in the background

When the analysis process finishes:
- Browse several of the results
- Expand the categories
- Double-click a broken analysis rule – what does this do?

Do **not** delete the result (one more workshop step – on the next slide)
UNIT

The IDz Workbench

Topics:
- Overview of Code Review Feature/Function
- Custom COBOL Code Review Development
- Running Code Review in Batch on z/OS

Note: There are two stakeholders needed to do custom Code Review development:

1. **Experienced Java programmer** – to code the rules using the IBM-supplied classes & wizards
2. **Experienced COBOL programmer** – to write the “spec” for each rule and test the solution
Creating a Custom COBOL Code Review rule - Introduction

To create user-written custom COBOL code review rules, you make use of IBM’s COBOL Application Model APIs. Using these APIs you can analyze selected COBOL language elements for user-written custom rules.

These APIs help parse COBOL programs and help us access all the statements and declarations, analyze them, and flag them as necessary.

The COBOL Application Model APIs consist of 2 APIs:

1. Custom Rules API for COBOL Code Review
   - The classes provided with this API let us manipulate the objects in the COBOL Application Model.
   - For instance, the *visit* method goes through a COBOL program’s syntax tree, and visits nodes of each type within the tree (for instance, a Paragraph node, a DISPLAY statement node, etc.). The nodes are supplied by the COBOL Application Model (CAM) API.

2. COBOL Application Model (CAM) API
   - The CAM API provides interfaces for accessing the individual elements of a COBOL program while the program is being analyzed by the COBOL Code Review API.
   - In other words, the CAM API parses a program and creates objects that can be manipulated to find information about all the COBOL statements and declarations.
   - You can override the *visit* method of the node that you want to analyze … for instance, the *DisplayStmt* node, and write our custom code to analyze all the display statements within the program.
Creating a Custom COBOL Code Review rule – Steps

1. Use the Plug-in Development Environment (PDE) wizard to:
   - Create the java plug-in project
   - Create a new category to hold all your domain specific rules
   - Add a rule to the category
   - Create a java class skeleton for your custom rule

2. Using the IDz COBOL Application Model API, fill in the skeleton with java code to implement your custom rule.

3. Test your custom rule
   - Create a new Eclipse Application Debug Configuration
   - Launch a new instance of IDz in Debug Mode
   - Add your new rule to a Software Analyzer configuration
   - Run a code review to test your rule.

4. Package your plugin as an update site and install it in the IDz Eclipse environment using Eclipse Software Updater.
Create the Java plug-in project – 1 of 3

- Open an instance of IDz
  - click the File menu, and click Other:

- Select the **Plug-in Project** option from the project creation wizard, and click **Next >**

- Name the project, and click **Next >**
Create the Java plug-in project – 2 of 3

- Specify custom properties for your plug-in project and click **Next >**

- When prompted, access the **Plugin Development** perspective
Create a new category to hold all your domain specific rules – 3 of 3

- Select: **Custom COBOL Code Review Rule** and click **Next >**

  ![Select Custom COBOL Code Review Rule]

- From the **COBOL Rule Template**:
  - Type in a Rule Label – this will appear as a rule violation annotation, when you run Code Review
  - Add a new Category – or select an existing category
  - Expand the window – and open the COBOL elements tree view
  - Check the statement(s) you wish to work with and click **Finish**

  ![COBOL Rule Template]

  … If prompted to switch Perspectives click **Yes**
Fill in the skeleton with java code to implement your custom rule.

Open your `<Rule>.java` file – under the `\src\<package>` - and edit the file, adding Java statements to build out your custom rule:

```java
package customrulesplugin;

import java.util.ArrayList;

public class CobolRule extends AbstractCustomCobolAnalysisRule {
    @Override
    public List&lt;ASTNode&gt; performRule(ASTNode baseNode) {
        final List&lt;ASTNode&gt; nodes = new ArrayList&lt;ASTNode&gt;();
        COBOLVisitorAdapter adapter = new COBOLVisitorAdapter();
        adapter.accept(baseNode, new AbstractCOBOLVisitor()) {
            @Override
            public void unimplementedVisitor(String s) {
            }
            @Override
            public boolean visit(ExecCicsStmt n) {
                if (n.getCicsStmt() instanceof CicsLinkStmt ||
                    n.getCicsStmt() instanceof CicsXCTLSstmt) {
                    nodes.add(n);
                }
                return true;
            }
        };
        return nodes;
    }
```
Specify the default rule severity

From the new plugin editor/view:
1. Select **Extensions** tab
2. Expand your rule
3. Open the drop-down box and select the default severity level, for your new rule

For most rules all you’ll want to do is customize the rule severity level. Use the drop-down to select: 0, 1, 2
Go to the **Run** Menu, and click **Debug Configurations**

- Click the **Eclipse Application** item, and then click the **New launch configuration** button ➔

- Name the Eclipse Application Debug Configuration and click **Close**
Test your custom rule - Launch a new instance of IDz in Debug Mode

1. From the **plugin.xml** view:
   - Launch an Eclipse application in Debug mode

2. From the **Overview** tab:

3. Click: Launch an Eclipse application in Debug mode

When the new instance of IDz opens for the first time you will need to create a Workspace:

1. New connection; 2. Connect to your host system; 3. Create a Property Group;
Test your custom rule - Add your new rule to a Software Analyzer configuration – 1 of 3

Using the steps from the previous slides, access a program and create a new Software Analyzer configuration that includes your custom rule:

1. From the Context Menu select Software Analyzer Configurations
2. Select: Software Analyzer and click: New launch configuration
3. Name your Software Analyzer configuration. Expand the COBOL Analysis Domains and Rules. You will see your:
   - New Custom Category
   - New Code Review rule – and it will have the severity level you set in a prior step

Click Close and save Yes

Name: Custom Rule Test
Test your custom rule - Add your new rule to a Software Analyzer configuration – 2 of 3

From the Context Menu, select: **Software Analysis** – and choose your new ruleset to run…
Test your custom rule - Add your new rule to a Software Analyzer configuration – 3 of 3

When code review completes, expand the Code Review results. Double-click your custom rule and it will select the source that violates it.

Close the temporary instance of IDz when you're finished.
Add addition custom COBOL rules ... or ... Export the plugin

When you return to instance of IDz you've been working in, you can create more custom COBOL code review rules. When you’re finished, simply export the plugin which can be installed in your team’s IDz client like any other standard Eclipse plugin.
Example Code Review rules

- A batch program should call a specific shop-standard ABEND routine if execution is stopped programmatically due to an error-condition.
- Use SEARCH ALL—with tables over 100 entries (unless the data cannot be sorted).
- All Files/Cursors with an OPEN statement must have a related:
  - A CLOSE statement
  - A FILE STATUS clause
- All variables within an arithmetic expression must be declared as PIC S9 (…) COMP or COMP-3
  - An example of checking for a Sign in a numeric field
    - This includes Perform…Varying, and Occurs Depending On field references.
- SQL statements should NOT:
  - Join more than <user defined number of> tables
  - Nest more than <user defined> levels in a sub-select
  - Use LIKE with a % or underscore in the first position
- Identify all the paragraphs that are not performed.
- GOTO paragraph names should follow standards based on the installation.
  - Ex: The GOTO paragraph should always go to the paragraph name suffixed with hyphen EXIT.
- Comment all DISPLAY and CALL statements.
Deeper dive on the Java – API Classes

- IDz provides ~1,000 APIs to parse your COBOL program nodes.  

Deeper dive on the Java – Taxonomy of a program

- Your COBOL program is a complex hierarchical structure (an abstract “tree”) of language elements – or in the CAM parlance… "nodes"

**Custom Rules API for COBOL Code Review**
- The classes provided with this API let us manipulate the objects in the COBOL Language Model.
- For instance, the *visit* method goes through a COBOL program’s syntax tree, and visits nodes of each type within the tree (for instance, a Paragraph node, a DISPLAY statement node, etc.).

**COBOL Application Model (CAM) API**
- The CAM API provides interfaces for accessing the individual elements of a COBOL program while the program is being analyzed by the COBOL Code Review API.
- In other words, the CAM API parses a program and creates objects that can be manipulated to find information about all the COBOL statements and declarations.
Annotated custom method

```java
package customplugin;

import java.util.ArrayList;

public class CobolRule extends AbstractCustomCobolAnalysisRule {

    @Override
    public List<ASTNode> performRule(ASTNode baseNode) {
        final List<ASTNode> nodes = new ArrayList<ASTNode>();
        COBOLVisitorAdapter adapter = new COBOLVisitorAdapter();
        adapter.accept(baseNode, new AbstractCOBOLVisitor());
        @Override
        public void unimplementedVisitor(String s) {
        }
        @Override
        public boolean visit(ExecCicsStmt n) {
            if (n.getCicsStmt() instanceof CicsLinkStmt ||
                n.getCicsStmt() instanceof CicsXCTLStmt)
            { nodes.add(n); }
            return true;
        }
    }
    return nodes;
}
```

Note: AST == “Abstract Syntax Tree”
Types of custom COBOL Rules – and the APIs

- Consider that there are three levels or types of custom COBOL rule requirements:
  - **Custom Parser Rules**
    - Complex Java coding.
    - Possibly custom parsing using Tokenizer class
  - **Complex API-based Rules**
    - Multiple Java methods
    - Utilize IDz CAM Nested APIs
  - **Simple API-based Rules**
    - Evaluate standalone elements (nodes) in a COBOL program for violations of coding standards
    - Evaluate related COBOL program nodes

**Examples:**
- Find DB2 Table Joins with more than 3 tables
- Find DB2/SQL statements with > 2 UNIONS
- COBOL paragraphs must be prefaced with a comment that references the paragraph name

**Examples:**
- All file I/O statements in the PROCEDURE DIVISION must reference the FILE STATUS variable
- All COBOL math statements must contain the ONSIZERRROR clause

**Numeric variables must be signed**

```java
@Override
public boolean visit(NumericItem n) {
    if (!n.isSigned()) {
        nodes.add(n);
    }
    return true;
}
```

**Sequential Files must reference File Status**

```java
public boolean visit(SequentialFileControlEntry n) {
    if (n.getStatus() == null) {
        nodes.add(n);
    }
    return true;
}
```
Alternate uses for Custom Software Analyzer/Code Review Rules

The LPEX/ISPF editor provides a number of useful source filters in the Context Menu:

- Check out the Filter view

While you don’t have the option to add to this list, you can create your own custom filtering using Software Analyze custom rules to isolate, and provide hyper-linked access to:

- IMS functionality
- DB2 functionality
- QSAM/VSAM functionality
- Etc.
Recommended Java Links for the Code Review APIs

- The Java Doc – for the Code Review APIs

- The Wizard provides Java “Visitor” design pattern – which is ideal for adding new methods to a hierarchy of classes
  - [http://www.objectmentor.com/resources/articles/visitor](http://www.objectmentor.com/resources/articles/visitor)

- The Abstract Syntax Tree is the base framework for many powerful tools of the Eclipse IDE, including refactoring, Quick Fix and Quick Assist. The Abstract Syntax Tree maps plain Java source code in a tree form
UNIT

The IDz Workbench

Topics:

- Overview of Code Review Feature/Function
- Custom COBOL Code Review Development
- Running Code Review in Batch on z/OS
Code Review…Review - Three separate options/workflows

Interactive (one program at a time)

- Code Review a single program at a time:
  - During Edit
  - Or from the Context Menu

Local Code Review –
- Still interactive mode, but can Code Review a PDS/library at a time
  - If the library has been copied down to a local workspace you can select and Code Review multiple programs

Batch –
- Code Review one or more PDS/libraries at a time...Or – Code Review specific members from a library in one run
  - The three Code Review Severity levels will return as MVS Job Step Completion Codes...
    - Allowing you to conditionally run other job steps – as part of Continuous Integration

MTEST.LIB.COBOL
Software Analyzer (Code Review) – Usage Model

Questions you need to answer:
- Will you run Code Review:
  - Interactively
  - In Batch
  - In Batch as part of Continuous Integration (DevOps)

Interactive Code Review
Software Analyzer (Code Review) – Batch Process

1. Verify Setup**
   - AKGCR Proc location and access
   - Java and USS files and environment variables
   - REXX Library
   - Obtain a run JCL sample that you can customize

2. Define Code Review/client files**
   - Rulesets
   - Property Group
   - z/OS File Map

3. Export Code Review/Client features to USS**

4. Customize JCL
   - Override Proc**
     - Specify COBOL/PL1 programs to analyze

5. Submit JCL and verify Job results

** One-time only step
1. Verify Setup

- **Proc (AKGCR)**
  - Must be accessible to job submit
  - Must specify correct //SYSPROC for %AKCGRX
    - The REXX Exec scans/analyzes the programs
  - Must specify correct for:
    - Java Home
    - IDzUTIL directory under USS
    - Can find correct values from RSED.ENVARs
    - Files exported to U.S.
      - Code Review Rules
      - Property Group file
      - z/OS File System Map

- Run JCL sample

```
CODEREVW.jcl

/*+---------------------------------------------------------------*/
//DDS0001A JOB MSGLEVEL=(1,1),MSGCLASS=A,CLASS=A
//PROC JCLLIB ORDER=(DDS0001.TEST.PROCLIB)
//AKGCREV EXEC PROC=AKGCR
//RULES DD PATH='/u/dds0001/cc/Test.dat'
//PROPERTY DD PATH='/u/dds0001/cc/TestPG.xml'
//MEMBERS DD *
//RTMTNT
//CSV DD SYSOUT=*,RECFM=VB,LRECL=2051
//XML DD SYSOUT=*,RECFM=VB,LRECL=2051
//MSGS DD SYSOUT=*,RECFM=VB,LRECL=2051

/**

After you have created & exported files to USS, you will customize the DD Statement overrides for client files
```
2. Define and Export Client Files

1. Create your code review ruleset
2. Export the ruleset to your PC to your PC
3. Define your Property Group
   - SYSLIB – for copybook references
4. Export the Property Group file to your PC
5. Define your z/OS File System Map
   - Only necessary if your z/OS dataset names are not mapped to the default File System Mapping entries
6. Export the z/OS File System Map to your PC
3. Export Client Files to z/OS UNIX

The easiest way to Export the Client Files needed for batch Code Review is to:

- Import the files from your PC to a local Workstation project
  - File ➔ Import ➔ General ➔ File System
  - Select the directory(s) and check the box for the exported:
    - Ruleset
    - Property Group
    - z/OS File System map
- Drag & Drop the files from your Local Workstation project directory to USS
  - Drag individual files
  - Or drag a folder
- Note that you can also Drag & Drop files from the Local filter in Remote Systems directly onto z/OS UNIX
4. Customize JCL

- Customize the run JCL and override the AKGCR Proc – pointing to the Unix folders and files you dragged over to Unix

```jcl
//DDS0001A JOB MSGLEVEL=(1,1),MSGCLASS=A,CLASS=A
//PROCS JCLLIB ORDER=(DDS0001.TEST.PROCLIB)

//AKGCREV EXEC PROC=AKGCR
//RULES DD PATH='/u/dds0001/cc/Test.dat'
//PROPERTY DD PATH='/u/dds0001/cc/TestPG.xml'
//PDS DD DISP=SHR, DSN=DDS0001.TEST.COBOL
//MEMBERS DD *

TRTMNT

//CSV DD SYSOUT=*,RECFM=VB,LRECL=2051
//XML DD SYSOUT=*,RECFM=VB,LRECL=2051
//MSGS DD SYSOUT=*,RECFM=VB,LRECL=2051
```

**Batch Code Review Run JCL**

- Override PATH= JCL/Proc parameters
- Specify program library
- Optionally list PDS members for analysis

//PDS Can only point to a single Library

Leave off the //MEMBERS – DD card if you want to scan all members of a PDS

Additional doc the AKGCR/Batch Code Review processing can be found here:


The USS **My Home** Filter maps to: `/u/<TSO ID>`
5. Submit the JOB – verify results

- When your job finishes, you will see a # of reports created as JES Spool Files.
- MSGS is the report that most closely resembles the interactive Code Review results.
- Also note that the return code (**U00002**) is based on the highest severity result thrown from Code Review.
5. Batch JOB Spool files - explained

- Which programs were analyzed
- Which Software Analyzer reports were run
- Comma delimited Code Review file – for Excel
- The Code Review analysis in XML format
- Entries used for debugging Code Review process
- Code Review formatted like the interactive reporting
- Another Code Review process debugging report
- TSO printout from the Job (Proc & DD statements)
Add Batch Code Review Options

- Can run Batch Code review:
  - That includes a post-processing script: For example … print the MSGS spool file
    - Requires:
      - Coding the post-processing script (typically a Unix Shell Script)
      - Adding a PMSGS DD statement to the Proc and Run JCL
  - Against Custom Code Review Rules
    - Requires:
      - Creating the custom Code Review rules (see prior section of this file)
      - Add (export) the plugin into your IDz client
      - Transfer the plugins (and plugin directory) to USS – specifically to a central plugins location
      - Add a CUSTOM environment variable to the ACGCRADD Proc – that points to the central plugins location on USS
        - Add a //STDENV DD * stream – that includes a references to your central plugins location on USS
        - Note that you may well need help from the Sysprogs responsible for IDz system components (like the installed Procs)
      - Either install the custom rule plugin to your IDz client or customize your <ruleset>.dat file
      - Add an override to the run JCL that references your custom ruleset file location on USS
  - Against Baselined Code Review results

- Note that Jon Gellin’s excellent white paper fully explains and documents the options and features above:
Running against a Baseline

Exports a Baseline of program code

Inputs Baseline to subsequent code reviews

Need to add two additional DD cards:

1. //BEXPORT – points to a zip file on Unix that contains date/time-stamped program source (establishes a baseline)
2. //BIMPORT – provides date/time-stamped program source for code reviews that are run after the baseline has been established
Additional usability options (requires IDz v9.1.1)

//SYSLIB and/or //LIST DD cards

//** SYSLIB  SYSLIB concatenation, optional () *
//** - when concatenating multiple DDs/DSNs/paths, all are used *
//** - PROPERTY is ignored when SYSLIB is specified *

SYSLIB can be used just like SYSLIB in a compile job, and you no longer need to export and transfer a property group.

//** LIST  list of source files to process, optional () *
//** - when concatenating multiple DDs/DSNs/paths, all are used *
//** - when using concatenation or inline data, line length of data is *
//**   limited to 1024 characters *
//** - PDS, MEMBERS and EXTMAP are ignored when LIST is specified *
//** - sample list format *
//**   #Select={Include eXclude}, I is default *
//**   #Language=language *
//**   #Member=member, * is default *
//**   #Datasetname=dsn *
//**   L=COBOL D=MY.DATASET.COBOL M=AKG*

LIST can be used to include/exclude any number of members from any number of PDS's, which provides filtering options and flexibility – along with simpler syntax.

Using a LIST file (or DD*) you no longer need PDS, MEMBERS, or EXTMAP.
LIST DD Card - Example

//LIST DD DSN=JON.LIST,DISP=SHR

... contents of JON.LIST ...
L=COBOL D=MY.DATASET.SRC M=AKG*
L=COBOL D=MY.DATASET.SRC M=AKG234
L=PLI D=MY.DATASET.SRC M=AKG345
L=COBOL D=MY.DATASET.SRC M=AKG456
L=COBOL D=MY.DATASET.SRC M=AKG789
L=COBOL D=MY.DATASET.SRC M=AKG012
L=PLI D=MY.DATASET.SRC M=AKG112
(Optional Workshop) **Run the Ruleset using a Baseline**

- Create a new file in your z/OS Unix Home directory named: dwbase.zip

- Add the //BEXPORT DD shown here.......... this job will create a baseline
  - Note that you can also add the //BIMPORT DD Card shown here – just comment it out for now

  ```
  000000 //CUSTRULE DD DUMMY
  000007 //PROPERTY DD DUMMY
  000008 //EXTMAP DD DUMMY
  000009 //BEXPORT DD PATH='/u/dds0001/dwbase.zip'
  000010 //BIMPORT DD PATH='/u/dds0001/dwbase.zip'
  000011 //SCRIPT DD DUMMY PATH='&PATH./samples/cr.post.sh'
  000012 //PROPERTY DD PATH='/u/dds0001/cc/TestPG.xml'
  000013 //DPS DD DSG SUB DSG:DDS0001 TEST CODE
  ```

- Submit the JCL and check the results in JES

- Uncomment/Add the //BIMPORT DD Card – and comment out the //BEXPORT card

  ```
  000000 //PROPERTY DD DUMMY
  000008 //EXTMAP DD DUMMY
  000009 //* BEXPORT DD PATH='/u/dds0001/dwbase.zip'
  000010 //BIMPORT DD PATH='/u/dds0001/dwbase.zip'
  000011 //SCRIPT DD DUMMY PATH='&PATH./samples/cc_post.sh'
  ```

- Submit the JCL and check the results in JES - this will run Code Review using the baseline - and if you haven't made changes to your program, you should get a zero-return code [0000]

- Change the code in your program (add a new line that ensure your changes will result in a Code Review problem
  - Note that you can simply run Code Review interactively to be sure)

- Submit the same Baselined JCL as above (with //BIMPORT uncommented) - this time IDz should pick up your new program version and find the Code Review error that occurs after the code's been Baselined
(Optional) **Run the Ruleset in batch** ("Headless" mode) – From a Windows Command Line


- The above link describes the steps showing how to run Code Review from a Windows machine – using a batch (command line) interface

---

**Procedure**

To run the sample script:

1. Export rule information from the software analysis configuration window to create a rule file.
2. Set up directories, data files, and source code files to use as parameter values.
3. Open a command console. On Linux, set the DISPLAY environment variable to a valid X server and display number.
4. Run the script by entering a command on a single line. The following command shows an example of how to start the sample script:

```
codereviewbatch
   -workspace c:\ws\workspace_0001
   -rulefile c:\rf\rule_file.dat
   -exportdir c:\cobol_export_file
   -directory c:\source_code
   -includefile c:\rf\include_file
   -outputfile c:\tm\output_information
   -debug
```

"Headless mode" supports code being reviewed from Local workstation projects and distributed SCMs – such as:

- Git
- RTC
Learn more about Software Analyzer/Code Review

1. **YouTube videos:**
   - **Overview of Code Review**
     - https://www.youtube.com/watch?v=enOrRw05DiY
     - https://www.youtube.com/watch?v=xEPI4PQ8WSs
   - **Custom COBOL Code Review rules development and implementation**
     - https://www.youtube.com/watch?v=O09ijULrmCU
   - **Code Review as DevOps Continuous Integration**
     - https://www.youtube.com/watch?v=0roH6cr703w

2. **White papers:**

3. **Online documentation:**